CHAPTER

11

WINDOWS XP PROFESSIONAL APPLICATION SUPPORT

After reading this chapter and completing the exercises, you'll be able to:

- ♦ Understand the runtime environments and application support in Windows XP Professional
- ♦ Deploy DOS, Win16, and Win32 applications
- ♦ Fine-tune the application environment for DOS and Win16 executables
- ♦ Understand how to assign and publish applications using Group Policy
- ♦ Address application compatibility issues

In this chapter, you encounter the pieces of the Windows XP Professional operating system that endow it with its outstanding power and flexibility. Its numerous runtime environments include limited support for DOS and 16-bit Windows applications, as well as more modern 32-bit Windows applications. Here, you'll have a chance to examine the various subsystems that Windows XP Professional provides to support DOS applications, plus 16- and 32-bit Windows applications, and understand how they work. You'll also have a chance to learn about Windows XP's mechanisms to ensure compatible operation of multiple applications on a single machine.

WINDOWS XP PROFESSIONAL SYSTEM ARCHITECTURE

Fundamentally, the Windows XP Professional operating system incorporates three primary components: the environment subsystem, Executive Services, and user applications (see Figure 11-1).

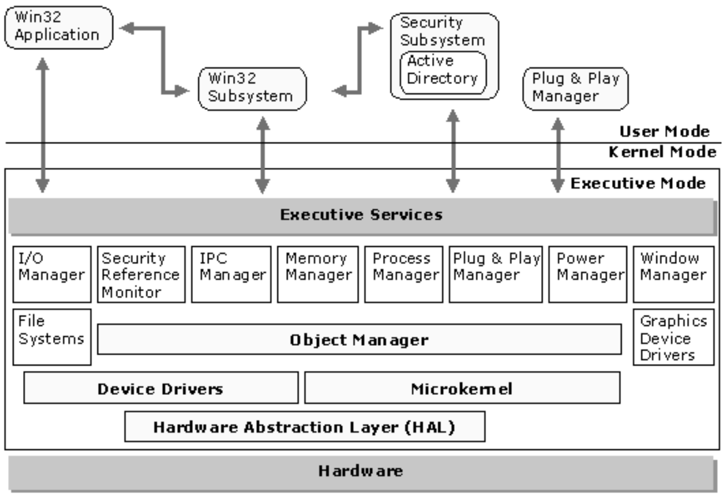


Figure 11-1 Components of the Windows XP Professional architecture

- **Environment subsystems** offer runtime support for a variety of different kinds of applications, under the purview of a single operating system. A **subsystem** is an operating environment that emulates another operating system (such as DOS or 16-bit Windows) to provide support for applications created for that environment, or a set of built-in programming interfaces that support the native Win32 (Windows 32-bit) runtime environment. Just like the applications they support, Windows XP Professional environment subsystems run in user mode, which means that they must access all system resources through the operating system's kernel mode.
- Windows XP Professional **Executive Services** and the underlying Windows XP **kernel** define the kernel mode for this operating system and its runtime environment. **Kernel mode** components are permitted to access system objects and resources directly, and provide the many services and access controls that allow multiple users and applications to coexist and interoperate effectively and efficiently.

- User applications provide the functionality and capabilities that rank Windows XP Professional among the most powerful network operating systems in use today. All such applications run within the context of an environment subsystem in Windows XP user mode. Applications and the subsystems in which they run have a mediated relationship because the client application asks the subsystem to perform activities for it, and the subsystem complies with such requests (or denies them if the requester lacks sufficient privileges).

To understand how these components fit together, we need to revisit the concept of processes and threads, building on concepts introduced in Chapter 1, "Introduction to Windows XP Professional."

Kernel Mode Versus User Mode

Before delving further into the architecture of Windows XP Professional, we'd better make clear the distinction between the Windows XP kernel mode and user mode. The main difference between the two modes lies in how memory is used by kernel-mode components and user-mode components.

In **user mode**, each process perceives the entire 4 GB of virtual memory available to Windows XP as its exclusive property—with the condition that the upper 2 GB of addresses are normally reserved for operating system use. This perception remains unaltered, no matter what kind of hardware Windows XP may run on. Note also that this address space is entirely virtual, and must operate within the confines of whatever RAM is installed on a machine and the amount of space reserved for the paging file's use. Although the upper limit for Windows XP virtual memory addresses may be 2 GB (or 4 GB, for system purposes), the real upper limit for Windows XP physical memory addresses will always be the sum of physical RAM size plus the amount of space in the paging file.

Although processes that operate in user mode may share memory areas with other processes (for fast message passing or sharing information), by default, they don't. This means that one user-mode process cannot crash another, or corrupt its data. This is what creates the appearance that applications run independently, and allows each one to operate as if it had exclusive possession of the operating system and the hardware it controls.

Note

If a user-mode parent process crashes, it will, of course, take its child processes down with it. (Parent and child processes are discussed later in this chapter.)

Processes running in user mode cannot access hardware or communicate with other processes directly. When code runs in the Windows XP kernel mode, on the other hand, it may access all hardware and memory in the computer (but usually through an associated Executive Services module). Thus, when an application needs to perform tasks that involve hardware, it calls a user-mode function that ultimately calls a kernel-mode function.

Because all kernel-mode operations share the same memory space, one kernel-mode function can corrupt another's data and even cause the operating system to crash. This is the reason that the environment subsystems contain as much of the operating system's capabilities as possible, making the kernel itself less vulnerable. For this reason, some experts voiced concern about the change in the Windows 2000 design that moved graphics handlers to the kernel. But because those graphics components were originally part of the Win32 environment subsystem—which must be available for Windows XP Professional to operate properly—a crash in either implementation could bring down the system. That's why this change has had little effect on the reliability or stability of Windows 2000 or Windows XP Professional.

Note

For a review of the user mode and kernel-mode architecture of Windows XP Professional, refer to Chapter 1.

Processes and Threads

From a user's point of view, the operating system exists to run programs or applications. But from the view of the Windows XP Professional operating system, the world is made of processes and threads. A **process** defines the operating environment in which an application or any major operating system component runs. Any Windows XP process includes its own private memory space, a set of security descriptors, a priority level for execution, processor-affinity data (that is, on a multiprocessor system, information that instructs a process to use a particular CPU), and a list of threads associated with that process. A list of currently active processes can be seen on the Processes tab of the Task Manager (see Figure 11-2). You access the Task Manager in any of several ways:

- Pressing Ctrl+Alt+Delete and clicking the Task Manager button (in normal Windows logon mode only)
- Pressing Ctrl+Alt+Delete (in Windows Welcome mode only)
- Right-clicking on an unoccupied area of the taskbar on your display, and selecting Task Manager from the resulting pop-up menu
- Pressing Ctrl+Shift+Esc

The basic executable unit in Windows XP is called a **thread**, and every process includes at least one thread. A thread consists of placeholder information associated with a single use of any program that can handle multiple concurrent users or activities. Within a multi-threaded application, each distinct task or any complex operation is likely to be implemented in its own separate thread. This explains how Microsoft Word, for instance, can perform spelling and grammar checks in the background while you're entering text in the input window in the foreground: two threads are running—one manages handling input, the other performs these checks.

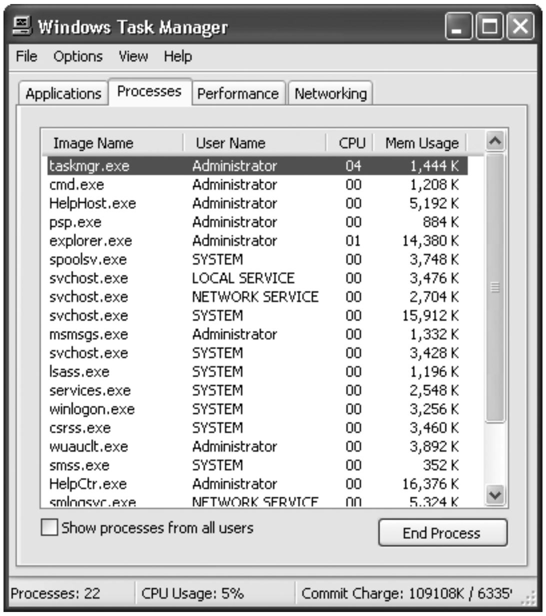

Windows Task Manager
File Options View Help
Applications Processes Performance Networking
Show processes from all users
End Process

Image Name	User Name	CPU	Mem Usage
taskmgr.exe	Administrator	04	1,444 K
cmd.exe	Administrator	00	1,208 K
HelpHost.exe	Administrator	00	5,192 K
psp.exe	Administrator	00	884 K
explorer.exe	Administrator	01	14,380 K
spoolsv.exe	SYSTEM	00	3,748 K
svchost.exe	LOCAL SERVICE	00	3,476 K
svchost.exe	NETWORK SERVICE	00	2,704 K
svchost.exe	SYSTEM	00	15,912 K
msmsgs.exe	Administrator	00	1,332 K
svchost.exe	SYSTEM	00	3,428 K
lsass.exe	SYSTEM	00	1,196 K
services.exe	SYSTEM	00	2,548 K
winlogon.exe	SYSTEM	00	3,256 K
csrss.exe	SYSTEM	00	3,460 K
wuauclt.exe	Administrator	00	3,892 K
smss.exe	SYSTEM	00	352 K
HelpCtr.exe	Administrator	00	16,376 K
smlogsvc.exe	NETWORK SERVICE	00	5,324 K

Processes: 22 CPU Usage: 5% Commit Charge: 109108K / 6335

Figure 11-2 The Process tab in Task Manager displays all currently active Windows XP Professional processes

Applications must be explicitly designed to take advantage of threading. Although it's safe to assume that most new 32-bit Windows applications—and the Windows XP operating system itself—are built to use the power and flexibility of threads, older 16-bit Windows and DOS applications are usually single-threaded. Also, it is important to understand that threads are associated with processes and do not exist independently. Processes themselves don't run, they merely describe a shared environment comprised of resources that include allocated memory, variables, and other system objects; threads represent those parts of any program that actually run.

Processes can create other processes, called **child processes**, and those child processes can inherit some of the characteristics and parameters of their **parent process**. (A child process is a replica of the parent process and shares some of its resources, but cannot exist independently if the parent is terminated.) The parent-child relationship between pairs of processes usually works as follows:

- When a user logs on to Windows XP Professional successfully, a shell process is created inside the Win32 subsystem within which the logon session operates. The Win32 subsystem is an operating environment that supports 32-bit Windows applications; this subsystem is required to run Windows XP. This process is endowed with a security token used to determine if subsequent requests for system objects and resources may be permitted to proceed. This shell process defines the Win32 subsystem as the parent process for that user.

- Each time a user launches an application or starts a system utility, a child process is created within the environment subsystem where that application or utility must run. This child process inherits its security token and associated information from the parent user account, but is also a child of the environment subsystem within which it runs. This "dual parentage" (security information from the user account and runtime environment from the environment subsystem) explains how Windows XP can run multiple kinds of applications in parallel, yet maintain consistent control over system objects and resources to which any user process is permitted access.

For example, each of the environment subsystems discussed in the following sections is an executable file—a combination of processes and threads running within the context of those processes (a **context** is the current collection of Registry values and runtime environment variables in which a process or thread runs). When an application runs in a Windows XP Professional subsystem, it actually represents a child of the parent process for the environment subsystem, but one that is endowed with the permissions associated with the security token of the account that launches the process. Whenever a parent process halts or is stopped, all child processes stop as well.

Environment Subsystems

Windows XP Professional offers support for various application platforms. Although primarily designed for 32-bit Windows applications, Windows XP Professional includes limited support for backward compatibility for 16-bit Windows and DOS applications.

Windows XP Professional's support for multiple runtime environments, also known as environment subsystems, confers numerous advantages, including:

- It permits users to run more than one type of application concurrently, including 32-bit Windows, 16-bit Windows, and DOS applications.
- It makes maintaining the operating system easier, because the modularity of this design means that changes to environment subsystems require no changes to the kernel itself, as long as interfaces remain unchanged.
- Modularity makes it easy to add or enhance Windows XP—if a new OS is developed in the future, Microsoft could decide to add a subsystem for that OS to Windows XP without affecting other environment subsystems.

The catch to using an architecture that supports multiple environment subsystems is in providing mechanisms that permit those subsystems to communicate with one another when necessary. In the Windows XP environment, each subsystem runs as a separate user-mode process, so that subsystems cannot interfere with or crash one another. The only exception to this insulation effect occurs in the Win32 subsystem: because all user-mode I/O passes through this subsystem, the Win32 subsystem must be running for Windows XP to function properly. If the Win32 subsystem's process ends, the whole operating system goes down with it. This explains why you can shut down processes associated with 16-bit Windows or DOS applications on a Windows XP machine without affecting anything other than those processes (and any related child processes) themselves.

Applications and the subsystems in which they run have a client/server relationship, in that the client application asks the server subsystem to do things for it, and the subsystem complies. For example, if a Win32 client application needs to open a new window (perhaps to create a Save As dialog box), it doesn't create the window itself, but asks the Win32 subsystem to draw the window on its behalf. If the 16-bit Windows on Windows environment is running and another 16-bit Windows application is launched, it will run within the existing 16-bit Windows environment by default.

The client issues the request through a mechanism known as a **local procedure call (LPC)**. The serving subsystem makes its capabilities available to client applications by linking them to a **dynamic link library (DLL)**. You could think of a DLL as a set of buzzers, where each one is labeled with the capabilities it provides. Pushing a specific buzzer tells the server subsystem to do whatever the label tells it to. This form of messaging is transparent to the client application (as far as it knows, it's simply calling a procedure). When a client pushes one of those buzzers (requests a service), it appears as if the act is handled by the DLL; no explicit communication with a server subsystem is needed. If a service isn't listed in the library, an application can't request it; thus, a word processor running in a command-line environment as a DOS application, for example, can't ask that subsystem to draw a window.

Message-passing is a fairly time-consuming operation, because any time the focus changes from one process to another, all the information for the calling process must be unloaded and replaced with the information for the called process. In operating system lingo, this change of operation focus from one process to another is called a **context switch**. To permit the operating system to run more efficiently, Windows XP avoids making context switches whenever possible. To that end, Windows XP includes the following efficiency measures:

- It caches attributes in DLLs to provide an interface to subsystem capabilities, so that (for example) the second time Microsoft Word requests a window to be created, this activity may be completed without switching context to the Win32 subsystem.
- It calls Executive Services (the collection of kernel-mode Windows XP operating system components that provides basic system services such as I/O, security, object management, and so forth) directly, to perform tasks without requesting help from an underlying environment subsystem. Because the kernel is always active in another process space in Windows XP, calling for kernel-mode services does *not* require a context switch.
- It batches messages so that when a server process is called, several messages can be passed at once—the number of messages has no impact on performance, but a context switch does. By batching messages, Windows XP allows a single context switch to handle multiple messages in sequence, rather than requiring a context switch for each message.

When LPCs must be used, they're handled as efficiently as possible. Likewise, their code is optimized for speed, and special message-passing functions can be used for different situations, depending (for example) on the size of the messages passed, or the circumstances in which they're sent.

So far, we've covered the broad view of how environment subsystems interact with client applications. Now, let's take a closer look at these subsystems.

The Win32 Subsystem

As the only subsystem required for the functioning of the operating system, the **Win32 subsystem** handles all major interface capabilities. In early versions of Windows NT, the Win32 subsystem included graphics, windowing, and messaging support, but since Windows NT 4.0 was released, these have been moved to the kernel and are now part of Executive Services. This applies equally to Windows 2000 and to Windows XP Professional.

In Windows XP, user-mode components of the Win32 subsystem consist of the console (text window support), shutdown, hard-error handling, and some environmental functions to handle such tasks as process creation and deletion. The Win32 subsystem is also the foundation upon which **virtual DOS machines (VDMs)** rest. These permit Windows XP to deliver both DOS and Win16 subsystems, so that DOS and Win16 applications can run on Windows XP unchanged (we'll talk more about VDMs and the DOS and Win16 subsystems later in this chapter). Try Hands-on Project 11-1 to launch a Win16 application in its own address space.

WIN32 APPLICATIONS

So far, we've examined the components of the Windows XP Professional operating system kernel. Now, it's time to see how applications run under that operating system.

The Environment Subsystem

As we've mentioned, the Win32 subsystem is the main environment subsystem under Windows XP, and the only one required for operation. Strictly speaking, even the other environment subsystems (the scaffolding that supports DOS and 16-bit Windows applications) are Win32 applications that run as child processes to the main Win32 process and support application environments called virtual DOS machines (VDMs) that run under Win32 to support DOS applications. (We explain VDMs and DOS application support in more detail later in this chapter.)

Multithreading

When a program's process contains more than one thread of execution, it's said to be a **multithreaded process**. The main advantage of multithreading is that it provides multiple threads of execution within a single memory space without requiring that messages be passed between processes or that local procedure calls be used, thus simplifying thread communication. Threads are easier to create than processes because they don't require as much context information, nor do they incur the same kind of overhead when switching from one thread to another within a single process.

Some multithreaded applications can even run multiple threads concurrently among multiple processors (assuming a machine has more than one). One more advantage to threading is that it's *much* less complicated to switch operation from thread to thread than to switch from one process to another. That's because every time a new process is scheduled for execution, the system must be updated with all the process's context information. Also, it's often necessary to remove one process to make room for another, which may require writing large amounts of data from RAM to disk for the outgoing process, before copying large amounts of data from disk into RAM to bring in the incoming process.

Note

As a point of comparison, a thread switch can normally be completed in somewhere between 15 and 25 machine instructions, whereas a process switch can take many thousands of instructions to complete. Because most CPUs are set up to handle one instruction for every clock cycle, this means that switching among threads is hundreds to thousands of times faster than switching among processes.

The big trick with multithreading, of course, is that the chances that one thread could overwrite another are increased with each additional thread, so this introduces the problem of protecting shared areas of memory from intraprocess thread overwrites. Windows XP manages access to memory very carefully, and limits which sections of memory any individual thread can write to by locking them, as you'll see in the next section. This largely avoids the problems associated with shared access to a single set of memory addresses.

Memory Space

Multithreaded programs must be designed so that threads don't get in each other's way, and they do this by using Windows XP **synchronization objects**. A section of code that modifies data structures used by several threads is called a **critical section**. It's very important that a critical section never be overwritten by more than one thread at once. Thus, applications use Windows XP synchronization objects to prevent this from happening, creating such objects for each critical section in each process context. When a thread needs access to a critical section, the following occurs:

1. A thread requests a synchronization object. If it is unlocked (not suspended in a thread queue), the request proceeds. Otherwise, go to step 2.
2. The thread is suspended in a thread queue until the synchronization object is unlocked for its use. As soon as this happens, Windows XP releases the thread and locks up the object.
3. The thread accesses the critical section.
4. When the thread is done, it unlocks the synchronization object so that another thread may access the critical object.

Thus, multithreaded applications avoid accessing a single data structure with more than one thread at a time by locking its critical section when it is in use and unlocking it when it is not.

Input Message Queues

One of the roles of the Win32 subsystem is to organize user input and get it to the thread to which that input belongs. It does this by taking user messages from a general input queue, and distributing them to the **input message queues** for the individual processes.

As we'll discuss later in this chapter, Win16 applications normally run within a single process, so they share a message input queue, unlike Win32 or DOS applications with their individual queues.

Base Priorities

When a program is started under Windows XP Professional, its process is assigned a particular priority class, generally Normal—but there is a range of options (see Figure 11-3). The priority class helps determine the priority at which threads in a process must run, on a scale from 0 (lowest) to 31 (highest). In a process with more than one active thread, each thread may have its own priority, which may be higher or lower than that of the original thread, but that priority is always relative to the priority assigned to the underlying process, which is known as the **base priority**. Managing priorities may be accomplished in one of several ways, and can sometimes provide a useful way to improve application performance. These include Task Manager and the Start command, as discussed in Chapter 10, "Performance Tuning."

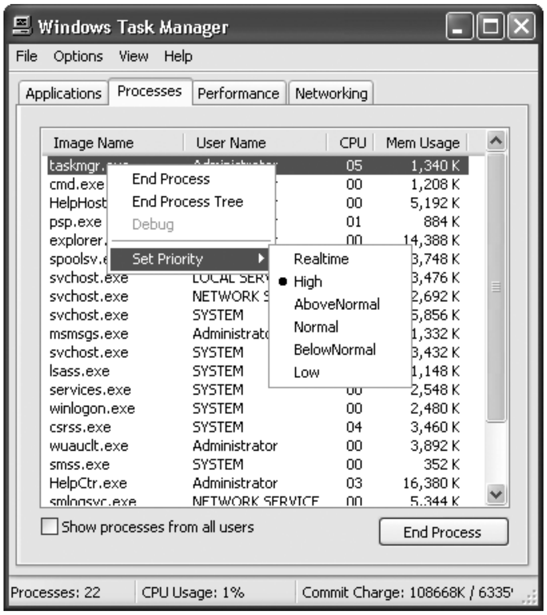


Image Name	User Name	CPU	Mem Usage
taskmgr.		05	1,340 K
cmd.exe		00	1,208 K
HelpHost		00	5,192 K
psp.exe		01	884 K
explorer.		00	14,388 K
spoolsv.e			3,748 K
svchost.exe	LOCAL SER		3,476 K
svchost.exe	NETWORK S		2,692 K
svchost.exe	SYSTEM		5,856 K
msmsgs.exe	Administrato		1,332 K
svchost.exe	SYSTEM		3,432 K
lsass.exe	SYSTEM		1,148 K
services.exe	SYSTEM	00	2,548 K
winlogon.exe	SYSTEM	00	2,480 K
csrss.exe	SYSTEM	04	3,460 K
wuauclt.exe	Administrator	00	3,892 K
smss.exe	SYSTEM	00	352 K
HelpCtr.exe	Administrator	03	16,380 K
smlogsvc.exe	NETWORK SERVICE	00	5,344 K

Processes: 22 | CPU Usage: 1% | Commit Charge: 108668K / 6335

Figure 11-3 The Task Manager's Process tab with priority options on display

DOS AND THE VIRTUAL DOS MACHINE

DOS and Win16 applications work somewhat differently from Win32 applications. Rather than each running in the context of its own process, these applications run within a virtual DOS machine (VDM), a special environment process that simulates a DOS environment so that non-Win32 Windows applications can run under Windows XP. In fact, it's reasonable to describe two separate operating environments that can run within a VDM: one supports straightforward DOS emulation and may be called the **DOS operating environment**; the other supports operation of Win16 applications within a VDM, and may be called the **Win16 operating environment**.

Any DOS operating environment under Windows XP occurs within a Win32 process named ntvdm.exe (see Figure 11-4). In fact, if you look at the Processes tab of the Task Manager when a DOS application is active, you'll see this process. The ntvdm process creates the environment wherein DOS applications execute. Each DOS application that is launched executes within a separate emulation environment. Thus, if you launch three DOS applications, three instances of ntvdm appear in the process list. Once a DOS application terminates, Windows XP also shuts down the emulation environment for that application by terminating the associated instance of ntvdm. This frees its system resources for re-use.

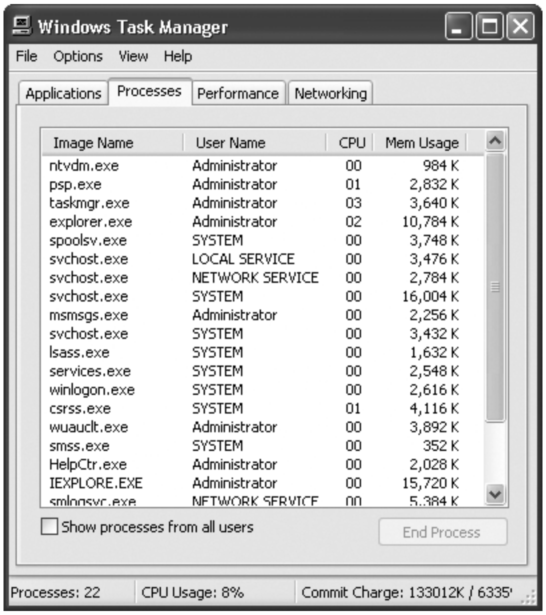


Image Name	User Name	CPU	Mem Usage
ntvdm.exe	Administrator	00	984 K
psp.exe	Administrator	01	2,832 K
taskmgr.exe	Administrator	03	3,640 K
explorer.exe	Administrator	02	10,784 K
spoolsv.exe	SYSTEM	00	3,748 K
svchost.exe	LOCAL SERVICE	00	3,476 K
svchost.exe	NETWORK SERVICE	00	2,784 K
svchost.exe	SYSTEM	00	16,004 K
msmsgs.exe	Administrator	00	2,256 K
svchost.exe	SYSTEM	00	3,432 K
lsass.exe	SYSTEM	00	1,632 K
services.exe	SYSTEM	00	2,548 K
winlogon.exe	SYSTEM	00	2,616 K
csrss.exe	SYSTEM	01	4,116 K
wuauclt.exe	Administrator	00	3,892 K
smss.exe	SYSTEM	00	352 K
HelpCtr.exe	Administrator	00	2,028 K
IEXPLORE.EXE	Administrator	00	15,720 K
smlogsvc.exe	NETWORK SERVICE	00	5,384 K

Processes: 22 | CPU Usage: 8% | Commit Charge: 133012K / 6335

Figure 11-4 The Task Manager's Processes tab shows ntvdm.exe running when a 16-bit DOS application is loaded

The environment created in a VDM is not the same as that available to Win32 applications. Instead, it is equivalent to the environment of Windows 3.x Enhanced mode, in which each DOS application has access to 1 MB of virtual memory, with 1 MB of extended memory and expanded memory if necessary.

By default, all DOS applications run in their own VDMs. By default, all Win16 applications share a single VDM (just as they do in "real Windows 3.x" environments).

VDM Components

The VDM runs using the following files:

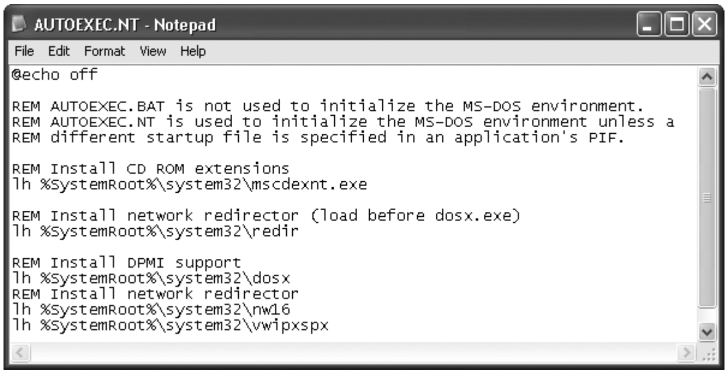
- *Ntio.sys*—The equivalent of Io.sys on MS-DOS machines, runs in **real mode** (real mode is a mode of operation for x86 CPUs wherein they can address only 1 MB of memory, broken into 16 64-KB segments). It provides "virtual I/O" services to the DOS or Win16 applications that run in a VDM.
- *Ntdos.sys*—The equivalent of Msdos.sys, runs in real mode. It provides basic DOS operating system services to the DOS or Win16 applications that run in a VDM.
- *Ntvdm.exe*—A Win32 application that runs in kernel mode. This is the execution file that provides the runtime environment within which a VDM runs. If you look at the list on the Processes tab of Task Manager, you'll see one such entry for each separate VDM that's running on your machine.
- *Ntvdm.dll*—A Win32 dynamic link library that runs in kernel mode. Ntvdm.dll provides the set of procedure stubs that fool DOS and Win16 programs into thinking they're talking to a real DOS machine with exclusive access to a PC, when in fact they're communicating through a VDM with Windows XP Professional.
- *Redir.exe*—The virtual device driver (VDD) redirector for the VDM. This software forwards I/O requests from programs within a VDM for I/O services through the Win32 environment subsystem to the Windows XP I/O Manager in Executive Services. Whenever a DOS or Win16 program in a VDM thinks it's communicating with hardware, it's really communicating with Redir.exe.

Virtual Device Drivers

DOS applications do not communicate directly with Windows XP drivers. Instead, a layer of **virtual device drivers (VDDs)** underlies these applications, and they communicate with Windows XP 32-bit drivers. Windows XP supplies VDDs for mice, keyboards, printers, and communication ports, as well as file system drivers (including one or more network drivers, each of which is actually implemented as a file system driver).

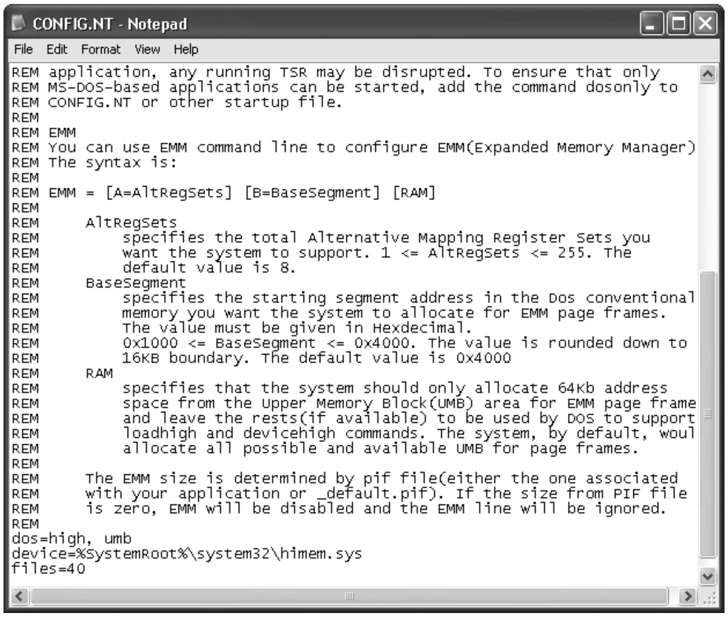
AUTOEXEC.BAT and CONFIG.SYS

When a DOS application is started, Windows XP runs the files specified in the application's program information file (PIF) or in AUTOEXEC.NT (see Figure 11-5) and CONFIG.NT (see Figure 11-6), the two files that replace AUTOEXEC.BAT and CONFIG.SYS for VDMs. AUTOEXEC.NT installs CD-ROM extensions and the network redirector. By default, Windows XP provides DOS Protected Mode Interface (DPMI) support, to permit DOS and Win16 applications to access more than 1 MB of memory within a virtual (or real) DOS machine. CONFIG.NT loads into an upper memory area for its VDM, and supports HIMEM.SYS by default to enable extended memory; it also sets the number of files and buffers available to DOS or Win16 programs, and provides necessary details to configure expanded memory.



```
@echo off

REM AUTOEXEC.BAT is not used to initialize the MS-DOS environment.
REM AUTOEXEC.NT is used to initialize the MS-DOS environment unless a
REM different startup file is specified in an application's PIF.

REM Install CD ROM extensions
lh %SystemRoot%\system32\mscdexnt.exe

REM Install network redirector (load before dosx.exe)
lh %SystemRoot%\system32\redir

REM Install DPMI support
lh %SystemRoot%\system32\dosx
REM Install network redirector
lh %SystemRoot%\system32\nw16
lh %SystemRoot%\system32\vwipxspx
```

Figure 11-5 AUTOEXEC.NT as it appears in Notepad



```
REM application, any running TSR may be disrupted. To ensure that only
REM MS-DOS-based applications can be started, add the command dosonly to
REM CONFIG.NT or other startup file.
REM
REM EMM
REM You can use EMM command line to configure EMM(Expanded Memory Manager)
REM The syntax is:
REM
REM EMM = [A=AltRegSets] [B=BaseSegment] [RAM]
REM
REM     AltRegSets
REM         specifies the total Alternative Mapping Register Sets you
REM         want the system to support. 1 <= AltRegSets <= 255. The
REM         default value is 8.
REM     BaseSegment
REM         specifies the starting segment address in the Dos conventional
REM         memory you want the system to allocate for EMM page frames.
REM         The value must be given in Hexdecimal.
REM         0x1000 <= BaseSegment <= 0x4000. The value is rounded down to
REM         16KB boundary. The default value is 0x4000
REM     RAM
REM         specifies that the system should only allocate 64Kb address
REM         space from the Upper Memory Block(UMB) area for EMM page frame
REM         and leave the rests(if available) to be used by DOS to support
REM         loadhigh and devicehigh commands. The system, by default, woul
REM         allocate all possible and available UMB for page frames.
REM
REM     The EMM size is determined by pif file(either the one associated
REM     with your application or _default.pif). If the size from PIF file
REM     is zero, EMM will be disabled and the EMM line will be ignored.
REM
dos=high, umb
device=%SystemRoot%\system32\himem.sys
files=40
```

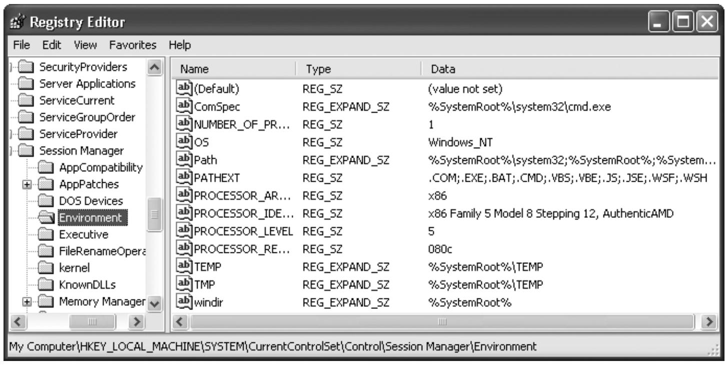
Figure 11-6 CONFIG.NT as it appears in Notepad

Try Hands-on Project 11-4 to explore AUTOEXEC.NT and CONFIG.NT.

Note

CONFIG.SYS isn't used at all by Windows XP, whereas AUTOEXEC.BAT is only used at system startup to set path and environment variables for the Windows XP environment. Neither file is consulted when it comes to running applications or initializing drivers; those settings must exist in the system Registry to work at all.

Once read from AUTOEXEC.BAT, path and environment variables are copied to the Registry, to HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Environment (see Figure 11-7).



Name	Type	Data
(Default)	REG_SZ	(value not set)
ComSpec	REG_EXPAND_SZ	%SystemRoot%\system32\cmd.exe
NUMBER_OF_PR...	REG_SZ	1
OS	REG_SZ	Windows_NT
Path	REG_EXPAND_SZ	%SystemRoot%\system32;%SystemRoot%;%System...
PATHEXT	REG_SZ	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
PROCESSOR_AR...	REG_SZ	x86
PROCESSOR_IDE...	REG_SZ	x86 Family 5 Model 8 Stepping 12, AuthenticAMD
PROCESSOR_LEVEL	REG_SZ	5
PROCESSOR_RE...	REG_SZ	080c
TEMP	REG_EXPAND_SZ	%SystemRoot%\TEMP
TMP	REG_EXPAND_SZ	%SystemRoot%\TEMP
windir	REG_EXPAND_SZ	%SystemRoot%

My Computer\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Environment

Figure 11-7 The Registry Editor shows the variables defined within the ...\Environment subkey

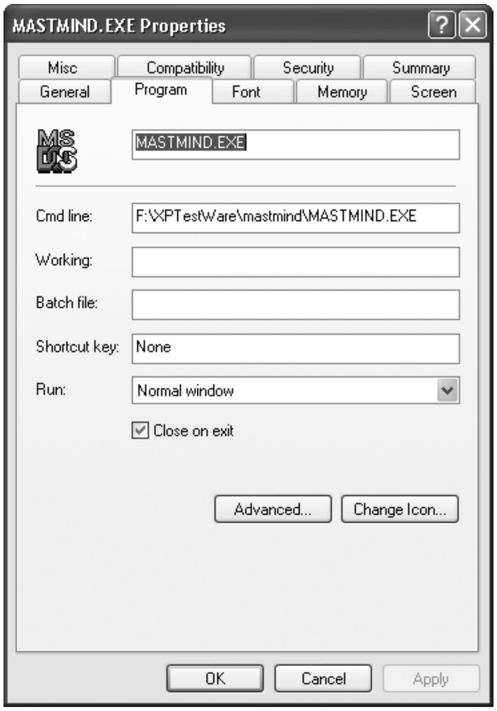
Custom DOS Environments

Windows XP offers customizable environment controls for its DOS runtime environment. These controls can be used to fine-tune or simply to alter how any DOS application functions. To customize a DOS application's execution parameters, open the Properties dialog box for that executable (.exe or .com) file. This is performed by right-clicking over an executable file and selecting Properties from the resulting menu. Try Hands-on Project 11-2 to explore the properties of DOS applications within Windows XP Professional.

The Properties dialog box for a DOS executable file has nine tabs. The General tab lists the same data items as any other file within the Windows XP Professional environment. The Program tab (see Figure 11-8) offers controls over:

- *Filename*—The name of the file
- *Command line*—Used to add command-line parameter syntax
- *Working*—Used to define the working directory, which is the directory from which the application will load files and where it saves files
- *Batch file*—Used to run a batch file before launching the executable file

- *Shortcut key*—Used to define a keystroke that launches the executable file
- *Run*—Used to define the window size of the DOS environment—normal, maximized, or minimized
- *Close on exit*—Informs the OS to close the DOS window when the application terminates
- *Advanced button*—Allows you to define the path to alternate AUTOEXEC.NT and CONFIG.NT files
- *Change icon button*—Changes the icon displayed for the executable file


MASTMIND.EXE Properties

Misc | Compatibility | Security | Summary
General | Program | Font | Memory | Screen

MASTMIND.EXE

Cmd line: F:\XPTestWare\mastmind\MASTMIND.EXE
Working:
Batch file:
Shortcut key: None
Run: Normal window
☑ Close on exit

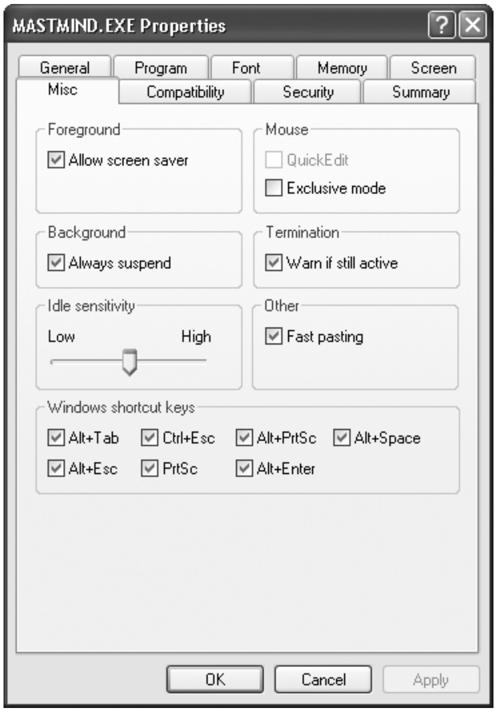
Figure 11-8 MASTMIND.EXE Properties, Program tab

The Font tab is used to define the font used by the DOS application. The Memory tab is used to define the memory parameters for the DOS environment that the corresponding ntvdm creates. These controls include settings for conventional memory, expanded memory (EMS), extended memory (XMS), and DOS protected-mode (DPMI) memory.

The Screen tab is used to define whether the DOS application loads full-screen or in a window. It also indicates if the ntvdm should emulate fast ROM, and whether or not it should allocate dynamic memory.

The Misc tab (see Figure 11-9) is used to define the following:

- Allow a screen saver over the DOS window
- Whether or not the mouse is used by the DOS application
- If the DOS application is suspended when in the background
- Whether or not to warn if the DOS application is active when you attempt to close the DOS window
- How long the application waits for I/O before releasing CPU control
- Whether or not to use fast pasting (a quick method for pasting information into the application; this doesn't work with some programs, so disable this checkbox if information does not paste properly)
- Which Windows shortcut keys are reserved for use by Windows XP Professional instead of the DOS application.



Foreground: [✓ Allow screen saver]

Mouse: [QuickEdit] [Exclusive mode]

Background: [✓ Always suspend]

Termination: [✓ Warn if still active]

Idle sensitivity: Low — High

Other: [✓ Fast pasting]

Windows shortcut keys: [✓ Alt+Tab] [✓ Ctrl+Esc] [✓ Alt+PrtSc] [✓ Alt+Space] [✓ Alt+Esc] [✓ PrtSc] [✓ Alt+Enter]

Figure 11-9 MASTMIND.EXE Properties Misc tab

The Compatibility tab provides access to modes of screen operation that are no longer available on the Settings tab in the Display applet. Here's where you can emulate screen behavior like that found in Windows 95, Windows 98, Windows NT 4, and Windows 2000 (using the Compatibility mode checkbox and its associated pull-down menu). You can also revert to older VGA emulation, including 640x480 resolution, 256 colors, and disable visual themes (none of which is accessible through the Display applet any more, either).

The Security tab provides access to standard user and group information, along with permissions data for each user and group named. The Summary tab provides access to file description and attribution information. This is primarily to provide better information for searching your file system for specific keywords or categories. For older files—especially DOS files—this kind of information is usually undefined.

Once you alter any portion of one of these tabs (except Security, which is a general-purpose NTFS file object control), a new shortcut for the application is created that retains whatever changes you make. Thus, you can reuse and fine-tune your custom DOS environment settings for each application, and even for multiple instances of the same application, if need be.

WIN16 CONCEPTS AND APPLICATIONS

Like DOS applications, Win16 applications also run in a VDM, although unlike DOS applications, which by default run in their own individual address spaces, all Win16 applications run in the same VDM unless you specify otherwise. This permits them to act like Win32 applications, and lets multiple Win16 applications interact with one another within a single VDM. This creates the appearance that multiple applications are active simultaneously. (Usually, only one Win16 application in a VDM can be active at any given moment, but this form of **multitasking**—which Microsoft calls cooperative multitasking—creates a convincing imitation of the more robust and real multitasking available to Win32 applications.) The **Win16-on-Win32 (WOW)** VDM runs as a multithreaded application, where each Win16 application occupies a single thread (see Figure 11-10), but where all such threads run by default in the context of a single VDM.

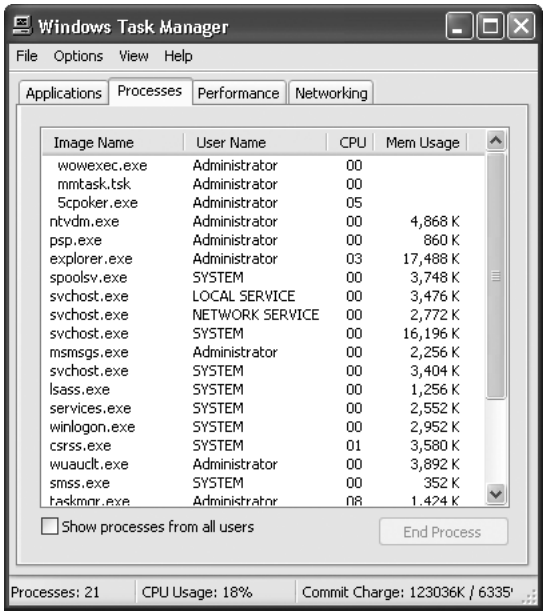


Image Name	User Name	CPU	Mem Usage
wowexec.exe	Administrator	00	
mmtask.tsk	Administrator	00	
5cpoker.exe	Administrator	05	
ntvdm.exe	Administrator	00	4,868 K
psp.exe	Administrator	00	860 K
explorer.exe	Administrator	03	17,488 K
spoolsv.exe	SYSTEM	00	3,748 K
svchost.exe	LOCAL SERVICE	00	3,476 K
svchost.exe	NETWORK SERVICE	00	2,772 K
svchost.exe	SYSTEM	00	16,196 K
msmsgs.exe	Administrator	00	2,256 K
svchost.exe	SYSTEM	00	3,404 K
lsass.exe	SYSTEM	00	1,256 K
services.exe	SYSTEM	00	2,552 K
winlogon.exe	SYSTEM	00	2,952 K
csrss.exe	SYSTEM	01	3,580 K
wuauclt.exe	Administrator	00	3,892 K
smss.exe	SYSTEM	00	352 K
taskmgr.exe	Administrator	08	1,424 K

☐ Show processes from all users

Processes: 21 | CPU Usage: 18% | Commit Charge: 123036K / 6335

Figure 11-10 The Task Manager Processes tab showing the wowexec environment

Win16-on-Win32 Components

The WOW VDM includes the following components:

- *Wowexec.exe*—Handles the loading of 16-bit Windows-based applications
- *Wow32.dll*—The dynamic link library for the WOW application environment
- *mmtask.tsk*—This is a multimedia background task module brought over from Windows Millennium edition (Me) into the Windows XP Professional environment, used to handle multimedia and video effects for graphical WOW applications. As Figure 11-10 indicates, this item shows up within the Wowexec.exe environment in Task Manager.
- *ntvdm.exe*, *ntvdm.dll*, *ntio.sys*, and *redir.exe*—Run the VDM
- *Vdmredir.dll*—The redirector for the WOW environment
- *Krnl386.exe*—Used by WOW on x86-based systems
- *Gdi.exe*—A modified version of Windows 3.x Gdi.exe
- *User.exe*—A modified version of Windows 3.x User.exe

Calls made to 16-bit drivers are transferred ("thunked") to the appropriate 32-bit driver without the application having to call that driver directly (or even know what's going on). Similarly, if a driver needs to return information to an application, it must be thunked back again. This back and forth translation helps explain why many Win16 applications run more slowly in a VDM on Windows XP, 2000, and Windows NT, than they do on other versions of Windows (even Windows 95), where no such translations are required.

Once a WOW environment is created, Windows XP sustains that environment until the system is rebooted or you manually terminate the Wowexec.exe task (through the Task Manager Processes tab). Creating new WOW environments each time a Win16 application is launched was deemed more costly in terms of resources and CPU time than maintaining a WOW environment once it had been created throughout a boot session. Thus, if you use Win16 applications often, this function offers you some modest performance benefits. But if you seldom use Win16 applications, you'd be better off terminating the WOW environment after you finish using the 16-bit application.

Memory Space

By default, all Win16 applications run as threads in a single VDM process (try Hands-on Project 11-5 to explore the number of threads used by a process). However, it might be a good idea not to permit this, because multiple threads running in a single process can affect the performance of each application. This mixture of applications can also make tracking applications more difficult, because most monitoring in Windows XP takes place on a per-process basis, not on a per-thread basis. Finally, running all Win16 applications in a single VDM means that if one of those applications goes astray and causes the VDM to freeze or crash, all applications in that VDM will be affected ("just like real Windows 3.x!").

Separate and Shared Memory

The "lose one, lose them all" effect of a single shared VDM explains why you might choose to run Win16 applications in separate VDMs. That way, you'll increase the reliability of those applications as a whole, and one errant application won't take down all the other Win16 applications if it crashes. Likewise, you'll make preemptive multitasking possible (that is, one busy application won't be able to hog the processor), and you'll be able to take advantage of multiple processors if you have them, because all the threads in a single VDM process must execute on the same processor.

The disadvantages of running Win16 applications in separate memory spaces hinge on memory usage and interprocess communications. Each additional process running on a machine requires about 2 MB of space in the paging file, and 1 MB of additional working set size; or the amount of data that the application keeps in memory at any given moment. Also, those older Win16 applications that don't support Dynamic Data Exchange (DDE) or Object Linking and Embedding (OLE) won't be able to communicate with each other if they run in separate VDMs.

It's also important to recognize that running Win16 applications as processes instead of threads increases the time it takes to switch from one application to another, because each switch requires a full context switch from one process to another. The best way to observe the impact of this separation is to try it the default way (wherein all Win16 applications share a single VDM), and then set up those Win16 applications in separate VDMs and compare the performance that results from each such scenario.

To launch a Win16 application in a separate memory space, you must first create a shortcut to the executable. Then edit the properties of the shortcut and select the *Run in separate memory space* checkbox. You can also start 16-bit applications in their own address spaces using the /separate command-line switch. The proper syntax is: start /separate [*16-bit program executable name*].

Note

The Run command in Windows XP Professional or Windows 2000 Professional does not have a *Run in separate memory space* checkbox like Windows NT 4.0 did; thus, a Win16 application cannot be launched in a separate memory space using the Run command.

Message Queues

As mentioned earlier, the Win32 subsystem is responsible for collecting user input and getting it to those applications that need it. However, unlike Win32 applications, all Win16 applications running in a single process share a message queue. Therefore, if one application becomes unable to accept input, it blocks all other Win16 applications in that VDM from accepting further input as well.

Threads

As mentioned earlier, Win16 threads that run in a VDM do not multitask like threads running in the Win32 subsystem. Instead of being preemptively multitasked, so that one thread can push another aside if its priority is higher, or so that any thread that's been taking up too much CPU time can be preempted, all application threads within a WOW VDM are cooperatively multitasked. This means that any one thread—which corresponds to any Win16 application—can hog the CPU. This is sometimes called the "good guy" scheduling algorithm, because it assumes that all applications will be well behaved and relinquish the CPU whenever they must block for I/O or other system services. The net effect, however, is that WOW VDMs behave as if they have only a single execution thread to share among all applications within that VDM.

Using Only Well-Behaved DOS and Win16 Applications

Many DOS applications, as well as numerous older Win16 applications, often take advantage of a prerogative of DOS developers—namely, the ability to access system hardware directly, bypassing any access APIs or drivers that the system might ordinarily put between an application and the underlying hardware. Although such applications work fine in DOS, Windows 3.x, and even Windows 98, this is not the case with Windows XP. The division into user mode and kernel mode in Windows XP means that any application that attempts to access hardware directly will be shut down with an error message to the effect of "illegal operation attempted."

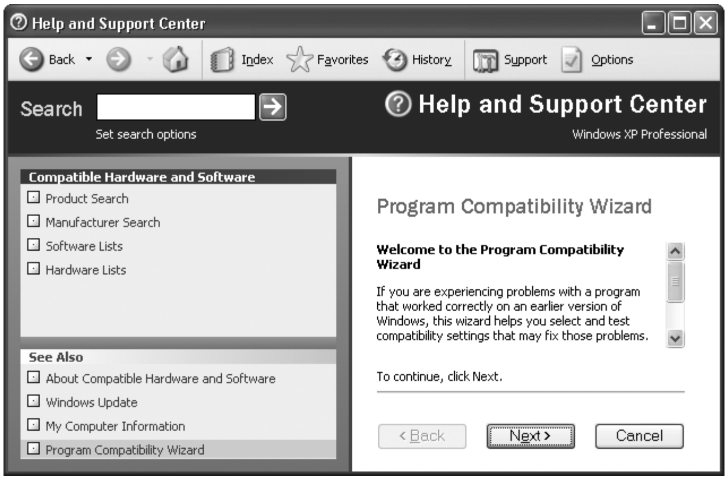
In Windows XP terminology, any application that attempts direct access to hardware is called "ill-behaved." Such applications will not run in a VDM. On the other hand, any Win16 or DOS application that uses standard DOS or Windows 3.x APIs instead of attempting direct access to hardware will work in a VDM. Such applications are called well behaved. Unfortunately, there is no list of well-behaved applications available, so the only way to tell the difference is to test the ones you'd like to use with Windows XP and see what happens. If an application doesn't perform properly, it shouldn't be deployed on your system. We suggest that you deploy only well-behaved applications for use with Windows XP, and that you seriously consider replacing any ill-behaved applications you may find in your current collection of programs. Try Hands-on Project 11-3 to explore the effects of VDMs in Windows XP.

Other Windows Application Management Facilities

Windows XP Professional supports additional methods for managing or accessing applications over and above what's been covered so far. These include a new Program Compatibility Wizard that supports installation of older applications that require APIs, DLLs, or other components from previous releases of Windows, and the ability to assign and/or publish applications using Group Policy objects from a Windows 2000 or .NET Server. Windows XP also offers an interesting and sophisticated method to resolve problems related to programs that use different versions of .DLLs with identical names, which was a source of difficult compatibility issues with earlier versions of Windows. These methods are covered in the following sections.

Program Compatibility Wizard

A new facility in Windows XP, the Program Compatibility Wizard is specifically designed to support the installation of older Windows applications that may occasionally cause problems, or fail to work altogether, when installed on Windows XP without the assistance of this tool. To access this tool, click the Program Compatibility link located at the bottom of the Compatibility tab of a program's Properties dialog box. Clicking the Start the Program Compatibility Wizard link launches the tool. This produces the Program Compatibility Wizard's welcome screen, as shown in Figure 11-11.



Program Compatibility Wizard

Welcome to the Program Compatibility Wizard

If you are experiencing problems with a program that worked correctly on an earlier version of Windows, this wizard helps you select and test compatibility settings that may fix those problems.

To continue, click Next.

Figure 11-11 The Program Compatibility Wizard starts with a welcome screen, then guides you through automated compatibility checks

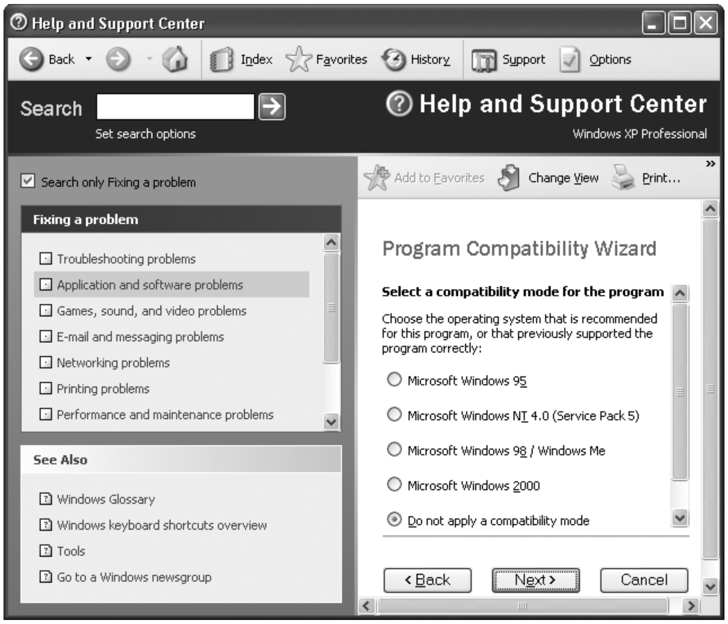
The next step in the process of working with this tool consists of selecting a single choice from a number of options to help manage compatibility issues for older applications, as follows:

- *I want to choose from a list of programs*—Produces a list of applications from a scan of your hard drives. From here, you can select any single program to proceed with the Wizard's assistance. The next choice is a compatibility mode setting, which we discuss later in this section; here, the point is that you can rely on an automated scan of your drives to show you which executable files might be in need of compatibility setting changes.
- *I want to use a program in the CD-ROM drive*—Points the automatic scan at a particular CD-ROM player in your machine instead of your hard drives, but otherwise works the same as the preceding option.
- *I want to locate the program manually*—Permits you to browse your hard drives to locate some particular program in a specific location. Thus, the Wizard provides a browse control, so you can select a single entry from the list of programs that resides in whichever directory you choose in response (we installed an old favorite, the Windows 3.11 version of Solitaire, Sol.exe).

After selecting a program, the Wizard displays a list of compatibility settings with radio buttons from which you can make a single selection, as depicted in Figure 11-12. The choices shown are as follows:

- Microsoft Windows 95
- Microsoft Windows NT 4.0 (Service Pack 5)
- Microsoft Windows 98/Microsoft Windows Me

- Microsoft Windows 2000
- Do not apply a compatibility mode



☑ Search only Fixing a problem

Fixing a problem

- Troubleshooting problems
- Application and software problems
- Games, sound, and video problems
- E-mail and messaging problems
- Networking problems
- Printing problems
- Performance and maintenance problems

See Also

- Windows Glossary
- Windows keyboard shortcuts overview
- Tools
- Go to a Windows newsgroup

Program Compatibility Wizard

Select a compatibility mode for the program

Choose the operating system that is recommended for this program, or that previously supported the program correctly:

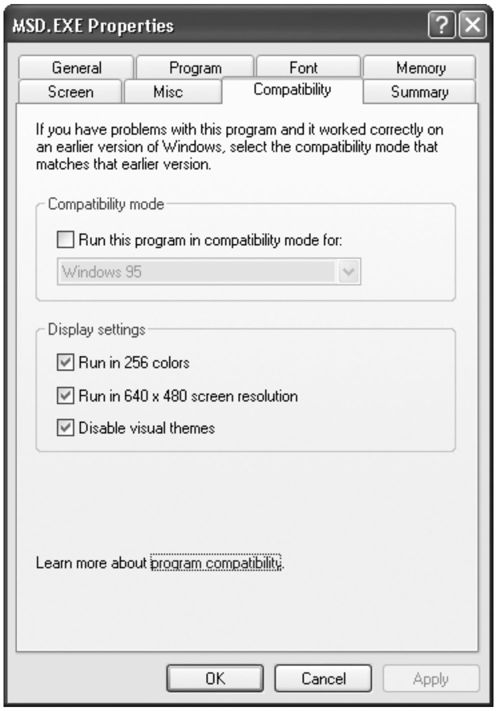
- ○ Microsoft Windows 95
- ○ Microsoft Windows NT 4.0 (Service Pack 5)
- ○ Microsoft Windows 98 / Windows Me
- ○ Microsoft Windows 2000
- [✓ Do not apply a compatibility mode]

Figure 11-12 Compatibility mode settings are selected from a specific list of available options

By default, the final element in the list—namely, Do not apply a compatibility mode—is selected. Unless you're sure which mode you need, you may have to follow a process of trial and error to determine which compatibility mode to use. The next screen in the wizard permits you to alter default display settings, which is often a requirement to make older programs work (because they were designed for environments where lower screen resolutions and reduced color levels were all that was available). This screen provides a set of checkboxes to permit you to limit the display to 256 colors, set resolution at strict 6402480 VGA resolution, and to disable Windows XP visual themes (so that older programs won't be adversely influenced by those settings).

Next in this process, the Wizard shows a screen that displays whatever settings you've chosen, as shown in Figure 11-13. It then gives you the option to test the program against whatever settings you've chosen. In our test case, we discovered that the Windows 95 compatibility mode does not work for the Windows 3.11 version of Solitaire, nor did any of the other settings available. Instead, we received an error message reading "The NTVDM CPU has encountered an illegal instruction," and the test window closed itself. Such experiences will be common for older applications that include illegal access to hardware (in other words, for ill-behaved DOS and Windows applications). A check of the Windows 95 version of Solitaire worked perfectly in Windows 95 compatibility mode.

Curiously, the version of Microsoft Diagnostics (msd.exe) included with Windows 3.11 ran perfectly without requiring any compatibility mode changes at all, though it did report numerous missing files and incorrectly reported drive sizes, and made numerous other errors that reflect the limitations inherent in Windows 3.11. Finally, it crashed when attempting to read the COM ports, presumably because it attempted illegal direct access to those ports. Thus, when you work with the Wizard, we urge you to test the default settings first, then work your way down the list of other options from the top down.



If you have problems with this program and it worked correctly on an earlier version of Windows, select the compatibility mode that matches that earlier version.

Compatibility mode

☐ Run this program in compatibility mode for:

Windows 95

Display settings

☑ Run in 256 colors

☑ Run in 640 x 480 screen resolution

☑ Disable visual themes

Learn more about program compatibility.

Figure 11-13 The Compatibility tab in the Properties window for any executable file provides direct access to the same controls offered through the Program Compatibility Wizard

In the next stage of the Wizard, the program may request permission to send the compatibility settings it establishes to Microsoft. You may choose to allow this information to be transmitted or not at your discretion. Whatever compatibility settings occur in the Wizard can subsequently be viewed and changed through the Compatibility tab in the program's Properties window. To access this window, right-click the program name or icon (depending on your current view in Windows Explorer/My Computer), then click the Compatibility tab. Figure 11-13 shows the settings we picked for MSD.EXE in the Wizard.

Note

The "Learn more about application compatibility" hyperlink on the Compatibility tab in the Properties window for any executable file takes you directly to the Help files on this subject.

Assigning and Publishing Applications on Windows XP Professional

In the Windows 2000 and Windows XP/.NET Server environments, you can use group policies to assign or publish programs to users or computers in a domain. It's especially useful to control access to programs through group memberships. Because it's easier to manage users in a group, rather than one at a time, this permits administrators to control access to applications by virtue of group management, thereby making it easy to establish which applications users can run by virtue of the groups to which they belong.

When you assign a program in the Windows environment, you're really assigning a Windows Installer package to some group or user. A Windows Installer package is nothing more than a complete set of software installation and configuration instructions. This provides an easy mechanism to manage access to software within an organization. When a user clicks an icon for an assigned application, the action automatically invokes the necessary instructions to install and configure the software on the computer where the user is working (assuming, of course, that he or she has the right group membership or user permissions to allow the request to proceed). Publishing a program requires some additional work, to properly construct the collection of files and instructions so that installation can occur upon demand.

Users who want to access assigned and published applications must belong to a Windows 2000 or Windows .NET Server domain. Of course, a domain administrator must also create an appropriate Microsoft Installer (MSI) package to handle the job as well. In fact, only someone with administrative privileges to a domain can configure a Group Policy Object for software assignment and publishing. The Microsoft Installer tool is also not included as part of the Windows XP Professional release, and it is most commonly invoked passively (and perhaps even unknowingly) by ordinary users. A file that contains instructions for the Windows Installer is called a **package**, and normally ends with the extension .msi. From an administrator's standpoint, it is necessary to create a network share that contains whatever Windows Installer packages (.msi files) are needed, plus any additional customizations that may apply to those basic .msi files, called **transforms** (.mst files), plus all necessary program files and related components.

Note

Additional information on working with the Windows Installer is covered in the discussion of IntelliMirror management technologies in Chapter 14. Numerous articles on the Windows Installer, and on building and deploying related packages, may be accessed in the Microsoft Knowledge Base, which is available online at *www.microsoft.com/kb/* or on the TechNet CDs. Also, an especially good white paper entitled "Windows Installer Service Overview" appears online at *http://www.microsoft.com/windows2000/techinfo/howitworks/management/installer.asp*.

To assign a program to a group within a Windows 2000 or .NET Server domain, it is necessary to construct the necessary .msi and .mst files (if needed), and to create a shared folder that contains those files plus all necessary program files and components with appropriate user and group permissions. From a logon with domain administrator privileges, creating the necessary Group Policy Object (GPO) occurs within the Active Directory Users and Computers MMC snap-in. From that point, follow these steps to assign a program to a group:

1. Select a directory container (domain, site, or organizational unit, a.k.a. OU) to which the GPO should be linked.
2. Create a new GPO for your MSI package; be sure to give that object a self-documenting name (e.g., "InstallOfficeXP").
3. Select the new GPO and click Edit to start the Group Policy snap-in so you can edit your new GPO.
4. Open, then right-click the Software Installation entry in the GPO, then click New Package.
5. When prompted for the path to the Windows Installer file (.msi), browse to the network share where that file resides; click the file, then click Open. When selecting files on a local hard drive, use a UNC name (e.g., \\computername\path); otherwise, other computers will look for the package on their local drives rather than on the machine where the share resides!
6. In most cases, you'll choose the Assigned item instead of the Advanced Published or Assigned option that's presented next (unless, as Microsoft points out, you have the necessary knowledge and need to use advanced options). This should create a software package element in the right-hand pane in the Group Policy snap-in.
7. From the Active Directory Users and Computers snap-in, select the container to which you linked your package GPO. Right-click the container, click Properties, then select the Group Policy tab. Next, click your new GPO, then click Properties.
8. Click the Security tab, and remove Authenticated Users from the list of assigned users and groups. Use the Add button to select whichever security group or groups to which you wish to grant access to the GPO, then set the access level to Read, then select the Apply Group Policy permissions.

Note

Changes to a GPO are not immediately imposed on affected computers; instead, they're applied during the next group policy refresh interval. If necessary, you can use the Secedit.exe command-line tool to impose such changes immediately; consult Microsoft Knowledge Base article Q277302 for the details on how to do so. To access the Knowledge Base online, please visit *http://search.support.microsoft.com/kb/c.asp*, or use the TechNet CDs.

Resolving DLL Conflicts in Windows XP

Windows XP includes a remarkable new technology called Windows Side by Side (WinSxS) isolation support. All versions of Windows have been prey to a problem that can occur when application installers blindly copy files onto a Windows machine without checking for potential compatibility problems. This can be particularly galling for a special kind of Windows code file called a dynamic link library (DLL), which is designed to be shared by multiple instances of the same program, or multiple programs, because it contains common interface objects, code elements, controls, and so forth. In many cases, different programs require different versions of the same DLLs, but, by default, whichever installer ran most recently will overwrite one version of a DLL necessary for some particular program with another version of the same DLL necessary for another particular program.

By default, Windows checks DLLs and other common code components before installing them on a computer. If it finds potential conflicts, it automatically makes the Registry modifications necessary to point to alternate versions of DLLs and other shared objects in a special directory named *%systemroot%*\WINDOWS\WinSxS. Then, when any program that requires a particular version of some DLL or other shared object runs, it automatically invokes the correct file to do its job properly.

This is a great improvement over earlier versions of Windows (including Windows 2000), where extensive analysis and comparison of DLL files was required, and hand-editing of a special Registry key named r1dllHell was also required to inform those versions of Windows that alternate versions of the files were to be used. Because this process is now completely automated in Windows XP Professional (and in Windows .NET Server), this once-vexing problem has been solved.

Note

For information on resolving DLL compatibility issues in other versions of Windows, consult the outstanding Knowledge Base article Q247957, entitled "Using DUPS to Resolve DLL Compatibility Problems" (DUPS stands for DLL Universal Problem Solver). You can access the Microsoft Knowledge Base online at *http://search.support.microsoft.com/kb/c.asp*, or on the TechNet CDs.

Chapter Summary

- Windows XP Professional is divided into three main parts: environment subsystems, Executive Services, and user applications. The environment subsystems provide support for applications written for a variety of operating systems, not just for Windows XP; the Executive Services define the Windows XP runtime environment; and user applications provide additional functionality for a variety of services, such as word-processing and e-mail applications.
- In addition to the basic Win32 Subsystem, two special-purpose operating environments (VDM and WOW) also run within that subsystem to provide limited backward compatibility for DOS and Win16 applications.

- Of these subsystems, only Win32 is crucial to the functioning of Windows XP as a whole. The other subsystems start up only as they're needed, but once launched, the WOW environment remains resident until the machine is shut down and restarted, or its parent VDM process is manually terminated within Task Manager.
- Windows XP includes some interesting additional application management facilities. The Program Compatibility Wizard may be used to manage compatibility modes and display settings for older Win16 or DOS applications. Specific Group Policy Objects to assign and publish Windows applications within Windows 2000 or Windows .NET Server domains may be configured and controlled, to permit members of groups with proper permissions to access and use such applications. Finally, Windows XP includes powerful, automated facilities to recognize and resolve potential conflicts with DLLs and other shared code objects without requiring user involvement.

Key Terms

base priority — The lowest priority that a thread may be assigned, based on the priority assigned to its process.

child process — A process spawned within the context of some Windows XP environment subsystems (Win32, OS/2, or POSIX) that inherits operating characteristics from its parent subsystem and access characteristics from the permissions associated with the account that requested it to be launched.

context — The collection of Registry values and runtime environment variables in which a process or thread is currently running.

context switch — The act of unloading the context information for one process and replacing it with the information for another, when the new process comes to the foreground.

critical section — In operating system terminology, this refers to a section of code that can be accessed only by a single thread at any one time, to prevent uncertain results from occurring when multiple threads attempt to change or access values included in that code at the same time.

DOS operating environment — A general term used to describe the reasonably thorough DOS emulation capabilities provided in a Windows XP virtual DOS machine (VDM).

dynamic link library (DLL) — A collection of virtual procedure calls, also called procedure stubs, that provide a well-defined way for applications to call on services or server processes within the Win32 environment. DLLs have been a consistent aspect of Windows since Windows 2.0.

environment subsystem — A mini-operating system running within Windows XP that provides an interface between applications and the kernel.

Executive Services — A set of kernel-mode functions that control security, system I/O, memory management, and other low-level services.

input message queue — A queue for each process, maintained by the Win32 subsystem, that contains the messages sent to the process from the user, directing its threads to perform a task.

kernel — The part of Windows XP composed of system services that interact directly with applications; it controls all application contact with the computer.

kernel mode — Systems running in kernel mode are operating within a shared memory space and with access to hardware. Windows XP Executive Services operates in kernel mode.

local procedure call (LPC) — A technique to permit processes to exchange data in the Windows XP runtime environment. LPCs define a rigorous interface to let client programs request services, and to let server programs respond to such requests.

multitasking — Sharing processor time between threads. Multitasking may be preemptive (the operating system may bump one thread if another one really needs access to the processor), or cooperative (one thread retains control of the processor until its turn to use it is over). Windows XP uses preemptive multitasking except in the context of the WOW operating environment, because Windows 3.x applications expect cooperative multitasking.

multithreaded process — A process with more than one thread running at a time.

package — The name of the collection of installer files, transforms, and other code components that support automated deployment of Windows programs. This term may also be applied to the .msi files associated with the Microsoft Installer facility used to drive automated installations through the Microsoft Installer itself.

parent process — The Windows XP environment subsystem that creates a runtime process, and imbues that child process with characteristics associated with that parent's interfaces, capabilities, and runtime requirements.

process — An environment in which the executable portion of a program runs, defining its memory usage, which processor to use, its objects, and so forth. All processes have at least one thread. When the last thread is terminated, the process terminates with it. Each user-mode process maintains its own map of the virtual memory area. One process may create another, in which case the creator is the parent process and the created process is the child process.

real mode — A DOS term that describes a mode of operation for x86 CPUs wherein they can address only 1 MB of memory, broken into 16 64-KB segments, where the lower ten such segments are available to applications (the infamous 640 KB), and the upper six segments are available to the operating system or to special application drivers—or, for Windows XP, to a VDM.

subsystem — An operating environment that emulates another operating system to provide support for applications created for that environment.

synchronization object — Any of a special class of objects within the Windows XP environment that are used to synchronize and control access to shared objects and critical sections of code.

thread — The executable portion of a program, with a priority based on the priority of its process—user threads cannot exist external to a process. All threads in a process share that process's context.

transform — A specific type of Microsoft Installer file that usually ends in .mst and that defines changes or customization to an existing Microsoft Installer package, and the .msi file in which the base installer instructions reside. Because most vendors (and Microsoft) define .msi files for their programs and systems, it's often easier to customize an existing .msi file with an .mst transform, rather than defining a new installer package from scratch.

user mode — Systems running in user mode are operating in virtual private memory areas for each process, so that each process is protected from all others. User-mode processes may not manipulate hardware, but must send requests to kernel-mode services to do this manipulation for them.

virtual device driver (VDD) — A device driver used by virtual DOS machines (VDMs) to provide an interface between the application, which expects to interact with a 16-bit device driver, and the 32-bit device drivers that Windows 2000 provides.

virtual DOS machine (VDM) — A Win32 application that emulates a DOS environment for use by DOS and Win16 applications.

Win16 operating environment — The collection of components, interfaces, and capabilities that permits Win16 applications to run within a VDM within the Win32 subsystem on Windows XP.

Win16-on-Win32 (WOW) VDM — The formal name for the collection of components, interfaces, and capabilities that permits the Win32 subsystem to provide native support for well-behaved 16-bit Windows applications.

Win32 subsystem — An operating environment that supports 32-bit Windows applications and is required to run Windows XP.

Review Questions

1. Which of the following is not an environment subsystem in Windows XP Professional? (Choose all correct answers.)
   a. Win32
   b. Windows on Windows (WOW)
   c. OS/2
   d. POSIX
2. If the threads in a process always runs on the same CPU in a multiprocessor system, that process is said to have a(n) ________________ for that processor.
3. Which of the following statements about process termination are true? (Choose all correct answers.)
   a. When a process's last thread is terminated, the process is terminated as well, unless it creates another thread within a certain interval.
   b. When a process terminates, all of its child processes terminate with it.
   c. A process must have at least one thread at all times.
   d. If a parent process terminates, its threads may be taken over by a child process.

4. Which of the following is not a reason to use the environment subsystem/kernel model?
   a. speed
   b. modularity
   c. subsystem protection
   d. ease of communication
5. The ________________ subsystem is required for the functioning of the Windows XP operating system.
6. Applications and the subsystems in which they run have a ________________ relationship, in that the client application asks the server subsystem to do things for it, and the subsystem complies.
7. When an application stops operating in user mode and begins operating in kernel mode, this is called a context switch. True or False?
8. Which of the following does not represent or result from an attempt to speed up subsystem/user application communications?
   a. LPCs
   b. caching services provided by the subsystem
   c. batching messages
   d. calling kernel services directly
9. Which two parts of the kernel were part of the Win32 subsystem prior to Windows NT 4.0?
   a. GDI
   b. I/O Manager
   c. device drivers
   d. Windows Manager
10. User applications always operate in user mode. True or False?
11. To access Windows program compatibility settings, you can: (Choose all correct answers.)
    a. Run the Program Compatibility Wizard
    b. Use the application compatibility tool (Apcompat.exe)
    c. Access the Compatibility tab in the program's Properties window
    d. None of the above
12. The Windows XP Executive Services belong to the kernel mode of this operating system and its runtime environment. True or False?
13. Windows XP Professional automatically detects and handles potential DLL conflicts. True or False?

14. Each time you start a DOS application under Windows XP, it runs in its own VDM. True or False?
15. Which of the following statements are true regarding LPCs, or Local Procedure Calls? (Choose all correct answers.)
    a. used to inform the CPU of I/O
    b. code optimized for speed
    c. supports specialized message passing functions
    d. employed only by the GDI portion of the OS
16. A child process can inherit the security token of its parent, or it can obtain a new security token by querying the Security Accounts Manager. True or False?
17. Windows 16-bit applications rely upon which of the following? (Choose all correct answers.)
    a. NTVDM
    b. POSIX
    c. WOW
    d. Win32
18. Under Windows XP, what do Win16 applications all share by default? (Choose all correct answers.)
    a. working directory
    b. message queue
    c. address space
    d. NTFS file permissions
19. Which of the following are true statements about Win16? (Choose all correct answers.)
    a. Wowexec.exe functions directly within a Win32 VM.
    b. When Wowexec.exe terminates, its NTVDM host also terminates.
    c. By default, all Win16 applications run within the same Wowexec.exe context.
    d. Only a single instance of Wowexec.exe can be launched.
20. Win16 applications can be launched into a separate memory space from the Run command within Windows XP. True or False?
21. In Windows XP, DOS applications can be launched into DOS environments with customized memory configurations. True or False?
22. All multithreaded applications running under Windows XP could be designed to operate on multiple processors for greater efficiency. True or False?
23. The first 16-bit application always runs in a separate memory space by default, whereas all subsequent 16-bit applications run by default in the same memory space as other applications of their kind. True or False?

24. A section of code that modifies data structures used by several threads is called a ________________.
25. Neither AUTOEXEC.BAT nor CONFIG.SYS has any role in determining Windows XP system configuration. True or False?

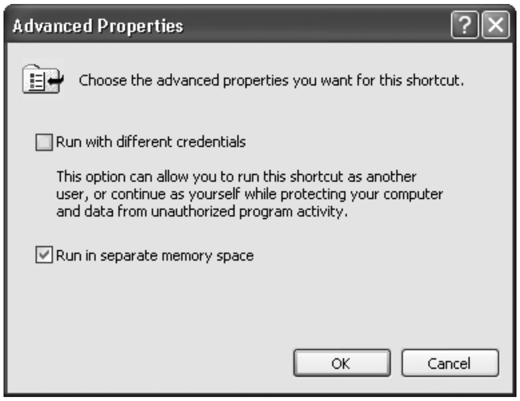
HANDS-ON PROJECTS

Project 11-1

To launch a Win16 application in its own address space:

1. Open Windows Explorer by selecting **Start|My Computer**.
2. In the right pane, double-click the drive letter that hosts your Windows XP main directory. If necessary, click **Tools|Folder Options,** and then on the **View** tab, click **Show hidden files and folders** to see the files in this volume. Click **OK**.
3. In the right pane, click the **Windows XP main folder** (this is WINDOWS by default). If necessary, repeat the instructions in the preceding step to view the files in this folder.
4. Scroll down in the right pane to locate Winhelp.exe (it may help to select the **Details** menu item in the **View** menu, rather than sticking with the default icon-based display).
5. Select **winhelp.exe**.
6. Right-click **winhelp.exe** and select **Create Shortcut** from the resulting menu.
7. Select **Shortcut to winhelp.exe**.
8. Right-click **Shortcut to winhelp.exe**, and select **Properties** from the resulting menu.
9. By default the Shortcut tab is selected. Click the Advanced button to open the Advanced Properties window, then click the **Run in separate memory space** checkbox (see Figure 11-14).

Note: Depending on your system settings, the file extensions may not appear.


Advanced Properties

Choose the advanced properties you want for this shortcut.

☐ Run with different credentials

This option can allow you to run this shortcut as another user, or continue as yourself while protecting your computer and data from unauthorized program activity.

☑ Run in separate memory space

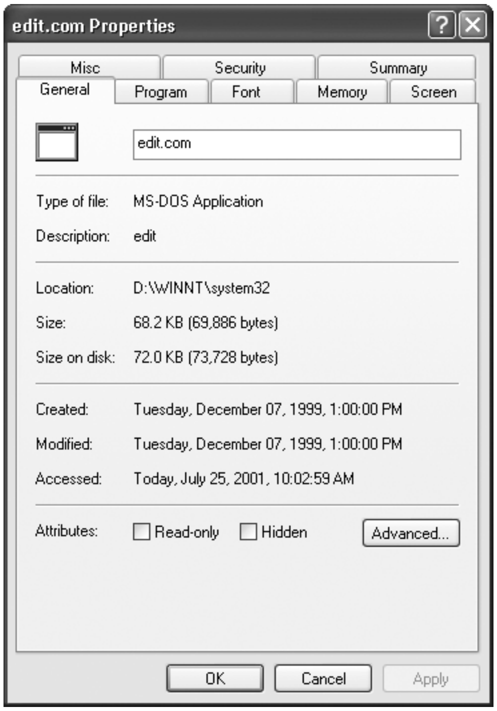
Figure 11-14 Configuring an application to run in a separate memory space from the Advanced Properties dialog box for a shortcut

10. Click **OK** twice to close the Advanced Properties window and apply the settings.
11. Double-click **winhelp.exe**. (This launches one instance of the 16-bit program.)
12. Double-click **Shortcut to winhelp.exe**. (This launches a second instance.)
13. Launch Task Manager by pressing **Ctrl+Shift+Esc**, then clicking the **Task Manager** button.
14. Select the **Processes** tab.
15. Notice that two WOW environments exist, each hosting an instance of Winhelp.exe.
16. Close Task Manager by selecting **File|Exit Task Manager**.
17. Close both instances of Windows Help by selecting **File|Exit**.

Project 11-2

To explore the Properties configuration for a DOS application:

1. Open Windows Explorer by selecting **Start|All Programs|Accessories|Windows Explorer**.
2. Locate the DOS application, edit.com. (Hint: you can use the Search function, selecting All files and folders as the search target, specifying edit.com as the file name to search for, then selecting Local Hard Drives as the search domain.)
3. Right-click **edit.com** and select **Properties** from the resulting menu.
4. View the details provided on the General tab (see Figure 11-15).



edit.com

Type of file: MS-DOS Application
Description: edit

Location: D:\WINNT\system32
Size: 68.2 KB (69,886 bytes)
Size on disk: 72.0 KB (73,728 bytes)

Created: Tuesday, December 07, 1999, 1:00:00 PM
Modified: Tuesday, December 07, 1999, 1:00:00 PM
Accessed: Today, July 25, 2001, 10:02:59 AM

Attributes: ☐ Read-only ☐ Hidden

Figure 11-15 The General tab of DOS application's properties

5. Select the **Program** tab and view the details provided.
6. Click the **Advanced** button and view the details provided.
7. Click **Cancel**.
8. Select the **Font** tab and view the details provided.
9. Select the **Memory** tab and view the details provided.
10. Select the **Screen** tab and view the details provided.
11. Select the **Misc** tab and view the details provided.
12. Explore other tabs in your Properties dialog box.
13. Click **Cancel** to close the Properties dialog box and discard any changes.

Project 11-3

To view the effects of various VDM-based applications on Windows XP:

1. Restart your Windows XP system and logon.
2. Launch Task Manager by pressing **Ctrl+Shift+Esc** and selecting **Task Manager** from the pop-up menu.

3. Select the **Processes** tab (refer to Figure 11-2).
4. Launch a DOS application, such as Edit.com, by using the **Run** command. This is performed by clicking the **Start** button then selecting **Run**. Type **c:\WINDOWS\system32\Edit.com** or a similar path to a DOS application. Click **OK**.
5. Notice in the list of processes through Task Manager that NTVDM has appeared, but the name of the DOS application itself has not.
6. Close the DOS application, using its own commands. For Edit.com this means selecting **File|Exit**.
7. Once the application terminates, notice that NTVDM no longer appears in the process list.
8. Launch a Windows 16 bit application, such as Winhelp.exe, by using the Run command. This is performed by clicking the **Start** button then selecting **Run**. Type **c:\windows\winhelp.exe** or a similar path to a Win16 application. Click **OK**.
9. Notice in the list of processes through Task Manager that NTVDM appears along with Wowexec.exe and Winhelp.exe as subitems.
10. Close the Win16 application using its own commands. For Winhelp.exe, this means selecting **File|Exit**.
11. Notice in the list of processes that NTVDM and WOWEXEC remain.
12. Terminate the WOWEXEC process by selecting it and clicking **End Process**.
13. Click **Yes** to confirm termination.
14. Notice that both NTVDM and WOWEXEC are no longer listed in the processes.
15. Close Task Manager by selecting **File|Exit Task Manager**.

Project 11-4

To view Windows XP's AUTOEXEC.NT and CONFIG.NT files:

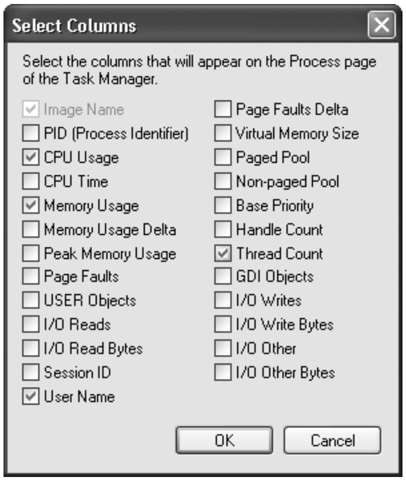
1. Launch Notepad by selecting **Start|All Programs|Accessories|Notepad**.
2. Select **File|Open**.
3. Change directories to **\WINDOWS\System32**.
4. Change the Files of type to **All Files**.
5. Locate and select **AUTOEXEC.NT**.
6. Click **Open**.
7. View the contents of this file (refer to Figure 11-5).
8. Select **File|Open**.
9. Change the **Files of type** to **All Files**.
10. Locate and select **CONFIG.NT**. (Hint: Type **c*.nt** into the File name: textbox to limit the list of files displayed to something close to the one you want!)

11. Click **Open** and view the contents of this file (refer to Figure 11-6).
12. Close Notepad by selecting **File|Exit**.

Project 11-5

To view the number of threads used by processes under Windows XP:

1. Launch Task Manager by clicking **Ctrl+Shift+Esc**.
2. Select the **Processes** tab (refer to Figure 11-2).
3. Select **View|Select Columns**.
4. Mark the **Thread Count** checkbox (see Figure 11-16).


Select Columns

Select the columns that will appear on the Process page of the Task Manager.

✓ Image Name	☐ Page Faults Delta
☐ PID (Process Identifier)	☐ Virtual Memory Size
✓ CPU Usage	☐ Paged Pool
☐ CPU Time	☐ Non-paged Pool
✓ Memory Usage	☐ Base Priority
☐ Memory Usage Delta	☐ Handle Count
☐ Peak Memory Usage	✓ Thread Count
☐ Page Faults	☐ GDI Objects
☐ USER Objects	☐ I/O Writes
☐ I/O Reads	☐ I/O Write Bytes
☐ I/O Read Bytes	☐ I/O Other
☐ Session ID	☐ I/O Other Bytes
✓ User Name	

Figure 11-16 To display Thread Count, check its associated checkbox in the Select Columns window

5. Click **OK**.
6. Maximize the Task Manager window.
7. Notice the number of threads for each of the currently active processes. (Note: you may have to resize the Task Manager display, or scroll to the right, to see this column; you can also click in the Threads entry in the columns above the display area to list processes ordered by thread count.)
8. Close Task Manager by selecting **File|Exit Task Manager**.

Project 11-6

To manage Compatibility Mode settings for Win16 applications in Windows XP Professional:

1. Obtain information from your instructor to copy the 16-bit version of MSD.EXE or some other executable to your local hard disk, in a directory of your choosing.
2. Use **My Computer** to navigate to the directory in which **MSD.EXE** resides. Right-click its name or icon, and choose **Properties** from the resulting pop-up menu.
3. Select the **Compatibility** tab in the Properties window. Notice that by default no operating system is selected in the Compatibility mode pane.
4. Check the following three checkboxes in the Display settings pane: **Run in 256 colors**, **Run in 6402480 screen resolution**, and **Disable visual themes**.
5. Click **OK** to accept your settings. Double-click the **MSD.EXE** name or icon to launch the program and observe the results. Compare the information reported here to that available from winmsd.exe (located in *%systemroot%*\WINDOWS\System32). Notice that the age of the program limits its ability to recognize large disk drives, newer CPUs, and that it crashes if you try to examine the COM ports. All of these behaviors reflect outmoded notions implemented as part of this program!

Note

Steps 7 and 8 are optional; if you want to conclude this project, skip directly to step 9.

6. Repeat steps 2 and 3. Uncheck the **Display Setting** checkboxes, then click **OK**. Re-launch **MSD.EXE** and observe the results.
7. Repeat steps 2 and 3. Check the checkbox in the **Compatibility mode** pane, select **Windows 95** as the compatibility target, and click **OK**. Re-launch **MSD.EXE** and observe the results.
8. Ask your instructor if he or she has any other older applications you can experiment with. Try working with various display settings and compatibility mode selections.
9. Close all open applications to complete this project.

Case Projects

1. To avoid the need to reimplement old code for your user community, which is in the process of upgrading from Windows 98 to Windows XP Professional, you decide to allow your users to run your company's homegrown application, Teller.exe, which is a well-behaved 16-bit Windows application, on their machines. Because this program sometimes hangs for as much as two or three minutes while computing end-of-day balances, it may cause problems for other 16-bit Windows applications that your users might need to run. What can you do to insulate these other applications from Teller.exe? How might you launch this program to accomplish this goal?
2. Provide a complete list of steps that you must complete to install two or more applications on a Windows XP machine, where such applications share a common DLL, but use different versions of that code. Explain exactly what you must do to create the proper Registry entries to resolve any potential DLL conflicts.
3. Given a list of DOS and 16-bit Windows applications that you may want to use on a Windows XP machine, what is the proper method to ensure that each of them will (or won't) work with this operating system? What happens if any of these applications is ill-behaved?